Swinburne University of Technology

Faculty of Science, Engineering and Technology

MIDTERM COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: Midterm, Convex Hull
Due date: May 2, 2021, 23:59
Lecturer: Dr. Markus Lumpe

Your name: Khang Trinh **Your student ID:** 102118468

Check Tutorial	Wed 08:30	Wed 10:30	Wed 16:30	Thurs 08:30	Thurs 10:30	Thurs 14:30	Thurs 16:30	Fri 08:30	Fri 10:30	Fri 14:30
					X					

Marker's comments:

Problem	Marks	Obtained
1	92	
2	138	
3	194 (buildConvexHull: 86)	
Total	424	

Problem 1

Vector2D.cpp

```
#include "Vector2D.h"
#include <math.h>

Vector2D::Vector2D(double aX, double aY) : fX(aX), fY(aY) {

}

void Vector2D::setX(double aX) {
	fX = aX;
}
double Vector2D::getX() const {
	return fX;
}
void Vector2D::setY(double aY) {
	fY = aY;
}
double Vector2D::getY() const {
	return fY;
}

Vector2D Vector2D::operator+(const Vector2D& aRHS) const {
	return Vector2D(fX + aRHS.getX(), fY + aRHS.getY());;
}

Vector2D Vector2D::operator-(const Vector2D& aRHS) const {
	return Vector2D(fX - aRHS.getX(), fY - aRHS.getY());
}

double Vector2D::magnitude() const {
	return sqrt(pow(fX, 2) + pow(fY, 2));
}

double Vector2D::direction() const {
	return atan2(fY, fX);
}

double Vector2D::dot(const Vector2D& aRHS) const {
	return fX * aRHS.getX() + fY * aRHS.getY();
}

double Vector2D::cross(const Vector2D& aRHS) const {
	return fX * aRHS.getY() - aRHS.getX() * fY;
}

double Vector2D::angleBetween(const Vector2D& aRHS) const {
	if (magnitude() == 0 && aRHS.magnitude() == 0)
		return 0;

	return acos(dot(aRHS) / magnitude() * aRHS.magnitude());
}

std::ostream& operator<<(std::ostream& aOutStream, const Vector2D& aObject) {
	aOutStream << "[" << aObject.fX << ", " << aObject.fY << "]";
	return aOutStream;
}

std::istream& operator>>(std::istream& aInStream, Vector2D& aObject) {
	double lX, lY;
	aInStream >> lX >> lY;
	aObject = Vector2D(lX, lY);
	return aInStream;
}
```

Problem 2

Point2D.cpp

```
#include "Point2D.h"
#include <iostream>

using namespace std;

static const Point2D gCoordinateOrigin;
static const double gEpsilon = 0.0001;

double Point2D::directionTo(const Point2D& aOther) const {
	return (*this - aOther).direction();
}

double Point2D::magnitudeTo(const Point2D& aOther) const {
	return (*this - aOther).magnitude();
}

Point2D::Point2D() : fId(""), fPosition(0, 0), fOrigin(&gCoordinateOrigin) {

}

Point2D::Point2D(const std::string& aId, double aX, double aY) : fId(aId),
fPosition(aX, aY), fOrigin(&gCoordinateOrigin) {

}

Point2D::Point2D(std::istream& aIStream) : fOrigin(&gCoordinateOrigin) {
	double lX, lY;

	aIStream >> fId >> lX >> lY;

	fPosition.setX(lX);
	fPosition.setY(lY);
}

const std::string& Point2D::getId() const {
	return fId;
}

void Point2D::setX(const double& aX) {
	fPosition.setX(aX);
}

const double Point2D::getX() const {
	return fPosition.getX();
}

void Point2D::setY(const double& aY) {
	fPosition.setX(aY);
}

const double Point2D::getY() const {
	return fPosition.getY();
}

void Point2D::setOrigin(const Point2D& aPoint) {
	fOrigin = &aPoint;
}

const Point2D& Point2D::getOrigin() const {
	return *fOrigin;
}

Vector2D Point2D::operator-(const Point2D& aRHS) const {
        return Vector2D(fPosition.getX() - aRHS.getX(), fPosition.getY() -
aRHS.getY());
}

double Point2D::direction() const {
        return directionTo(*fOrigin);
}

double Point2D::magnitude() const {
        return magnitudeTo(*fOrigin);
}

bool Point2D::isCollinear(const Point2D& aOther) const {
        double lResult = abs(direction() - aOther.direction());
        return lResult <= gEpsilon && lResult >= 0 || lResult <= 3.1416 && lResult >=
3.1415;
}

bool Point2D::isClockwise(const Point2D& aP0, const Point2D& aP2) const {
        return Vector2D(*this - aP0).cross(Vector2D(aP2 - aP0)) > 0;
}

bool Point2D::operator<(const Point2D& aRHS) const {
        Vector2D lResult = *this - aRHS;
        if (lResult.getY() <= -gEpsilon || lResult.getY() == 0 && lResult.getX() <= -
gEpsilon)
                return true;
        return false;
}

std::ostream& operator<<(std::ostream& aOStream, const Point2D& aObject) {
        aOStream << aObject.fId << ": (" << aObject.fPosition.getX() << ", " <<
aObject.fPosition.getY() << ")";
        return aOStream;
}

std::istream& operator>>(std::istream& aIStream, Point2D& aObject) {
        aObject = Point2D(aIStream);
        return aIStream;
}
```

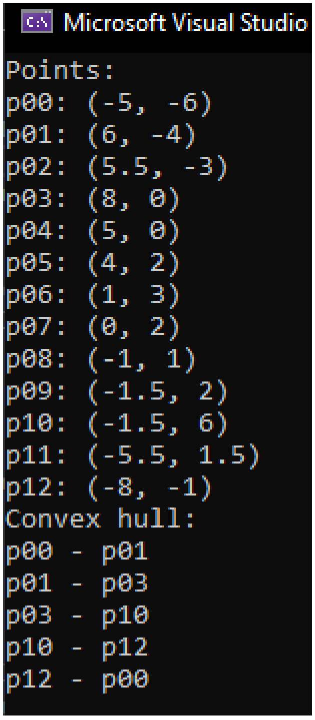
Problem 3

Point2DSet.cpp

```
#include "Point2DSet.h"
#include <fstream>
#include <algorithm>

using namespace std;
using Iterator = std::vector<Point2D>::const_iterator;

static const double gEpsilon = 0.0001;

void Point2DSet::add(const Point2D& aPoint) {
       fPoints.push_back(aPoint);
}

void Point2DSet::add(Point2D&& aPoint) {
       fPoints.push_back(aPoint);
}

void Point2DSet::removeLast() {
       fPoints.pop_back();
}

bool Point2DSet::doesNotTurnLeft(const Point2D& aPoint) const {
       return aPoint.isClockwise(fPoints[size() - 2], fPoints[size() - 1]);
}

void Point2DSet::populate(const std::string& aFileName) {
       int lPointCount;
       Point2D lPoint2D;

       ifstream aInStream(aFileName, ifstream::in);
       aInStream >> lPointCount;
       for (int i = 0; i < lPointCount; i++)
       {
              aInStream >> lPoint2D;
              add(lPoint2D);
       }
}

bool orderByCoordinates(const Point2D& aLeft, const Point2D& aRight) {
       return aLeft < aRight;
}

bool orderByPolarAngle(const Point2D& aLHS, const Point2D& aRHS) {
       if (aLHS.isCollinear(aRHS)) {
              return aLHS.magnitude() - aRHS.magnitude() <= -gEpsilon;
       }

       return aLHS.direction() - aRHS.direction() <= -gEpsilon;
}

void Point2DSet::sort(Comparator aComparator) {
       stable_sort(fPoints.begin(), fPoints.end(), aComparator);
}

void Point2DSet::buildConvexHull(Point2DSet& aConvexHull) {
       //Sort by Coords
       sort(orderByCoordinates);

       //Asign new Origin
       for (Point2D& point2D : fPoints)
       {
              point2D.setOrigin(fPoints[0]);
       }
```

```
    //Sort by Polar Angle
    sort(orderByPolarAngle);

    //Add first 3 points
    for (size_t i = 0; i < 3; i++)
    {
        aConvexHull.add(move(fPoints[i]));
    }

    //Graham Scan
    for (size_t i = 3; i < size(); i++)
    {
        while (aConvexHull.doesNotTurnLeft(fPoints[i]))
            aConvexHull.removeLast();
        aConvexHull.add(move(fPoints[i]));
    }
}

size_t Point2DSet::size() const {
    return fPoints.size();
}

void Point2DSet::clear() {
    fPoints.clear();
}

const Point2D& Point2DSet::operator[](size_t aIndex) const {
    return fPoints[aIndex];
}

Iterator Point2DSet::begin() const {
    return fPoints.begin();
}

Iterator Point2DSet::end() const {
    return fPoints.end();
}
```

<u>Output</u>



```
Points:
p00: (-5, -6)
p01: (6, -4)
p02: (5.5, -3)
p03: (8, 0)
p04: (5, 0)
p05: (4, 2)
p06: (1, 3)
p07: (0, 2)
p08: (-1, 1)
p09: (-1.5, 2)
p10: (-1.5, 6)
p11: (-5.5, 1.5)
p12: (-8, -1)
Convex hull:
p00 - p01
p01 - p03
p03 - p10
p10 - p12
p12 - p00
```